**说明文档**

1. **项目说明**

本项目实现了一个计算图，支持大作业第一阶段的所有要求。

1. **文件说明&依赖关系**

**Node.h & Node.cpp**

实现Node基类

**Scalar.h & Scalar.cpp** 依赖**Node.h**

实现三种标量类型的结点所对应的类：Variable，Placeholder，Constant

**Operator.h & Operator.cpp** 依赖**Node.h**

实现各种运算符所对应的类，包括：

对应算数运算的：

PlusOperator，MinusOperator，MultipleOperator，DivisionOperator类

对应函数运算的：

SinOperator，ExpOperator，LogOperator，TanhOpeartor，SigmoidOperator类

对应比较运算的：

GreaterOperator，GreaterEqualOperator，LessOperator，LessEqualOperator，EqualOperator类

对应两种特殊结点的：

PrintOperator，CondOperator类

**ComputationalGraph.h & ComputationalGraph.cpp** 依赖**Node.h**

实现ComputationalGraph类

**main.cpp** 依赖**Scalar.h**，**Operator.h**和**ComputationalGraph.h**

测试代码，用于测试ComputationalGraph类能否满足要求

**makefile**

用于Linux系统的makefile文件

**makefile\_win**

用于Windows系统的makefile文件，使用方法见readme.txt

1. **继承关系&接口说明**
2. **继承关系说明**

Node基类是抽象类，提供一个包含结点通用成员和接口的模板。

各种运算符所对应的类和三种标量类型对应的类均以public方式派生自Node基类。

ComputationalGraph类没有父类和派生类。

1. **接口说明**

以下为Node基类提供的接口：





对于Node的派生类，接口与Node类似，但构造函数的参数有不同。

以下为ComputationalGraph的接口：



1. **设计思路**

将计算图中所有种类的结点抽象为一种统一的基类，这个结点基类包含了指向其前置结点的指针，从而可以仅依靠结点就完成绝大多数计算。从结构上来说，“图”仅由结点和结点中的指针构成。

ComputationalGraph本身则主要提供对结点的访问方式，并提供一些需要实现但无法由结点完成的功能。

1. **实现细节**
2. **计算图的存储**

在ComputationalGraph类中，有一个名为NodeMap的map负责存储结点名称和结点地址的对应关系。

添加一个新的标量结点的时候，我们创建一个对应类型的对象。在对象初始化时用已知信息初始化对应的结点，然后将该结点的地址加入NodeMap中。

添加运算符结点的时候，与添加标量结点类似地，我们将对应运算符的对象的地址加入到NodeMap中；并将该运算符的计算所需要的结点的地址按顺序存储到Node下的Pre容器中。

1. **结点数值的计算**

计算图的计算通过计算图类下的Calc函数进行。通过vector参数传入初始情况，用vector中的赋值信息更新NodeMap中的信息，然后对要计算答案的结点进行操作。调用对应结点中的Calc函数，作为第一层递归。对于每一个结点、变量、常量和操作符、运算符，我们都在类中编写了对应的Solve函数，通过简单地递归调用结点下的Calc函数与相应的Solve函数即可得到结果。

1. **时间标记**

为了保证一个结点只被计算一次，我们引入了时间标记。在计算图中的时间标记在每次计算图开始计算前加一，如果计算到某个结点的时间标与当前计算图的时间标相等，说明该结点在本次计算图的运算过程中已经被计算过，那么直接返回其值即可。如果没有计算过，那么在对该结点计算前更新其时间标。

1. **错误信息（Error）的传递**

在计算计算图时出现的错误信息将保存在ErrorSignal中并立即返回上一层函数，ErrorSignal非空会使函数不断返回，直到回到计算图类下的Calc函数，在Calc函数中输出错误信息并将存储在PreAnswer中的答案置为零。每次进行计算图计算前会将ErrorSignal清空。

1. **测试代码&使用说明**

测试代码main.cpp大致展示了ComputationalGraph类是如何支持大作业第一阶段的要求的，并提供了ComputationalGraph类的使用方法的具体例子。

1. **添加结点**

以加入一个名字为x1，值为x的Constant类型标量为例：



以加入一个名字为x1，前置结点名字为x3的对应Log操作的结点为例：



1. **EVAL操作**

使用一个成员为pair的vector记录这次EVAL中需要被赋值的Placeholder的名字和对应的值，然后和需要用的输出流一起传入Calc函数。

Calc函数会返回一个值，但是输出在函数内已经完成了，所以用一个临时变量存储，以防将来要用到。



1. **Variable相关操作**

以SETANSWER操作为例，直接在对应的结点上做修改。



1. **优点**
2. **源代码的依赖关系少**

ComputationalGraph仅仅依赖于Node基类，和Scalar，Operator是独立的，从而降低了代码的耦合度。

1. **抽象度高**

本项目中大部分的接口，以及Node基类本身，都是高度抽象且灵活的。可以在不同的应用场景下起到不同作用。

这也带来了一定的开发上的便利性，例如在添加新的结点类型时，通常只需要实现新结点类型的构造函数和Solve函数。

1. **接口丰富**

对于绝大部分的类的私有成员，都提供了访问它们的接口，便于后续的开发。